**WEB前端处理跨域问题的8种解决方案**

**1.同源策略如下：**

|  |  |  |
| --- | --- | --- |
| **URL** | **说明** | **是否允许通信** |
| http://www.a.com/a.js http://www.a.com/b.js | 同一域名下 | 允许 |
| http://www.a.com/lab/a.js http://www.a.com/script/b.js | 同一域名下不同文件夹 | 允许 |
| http://www.a.com:8000/a.js http://www.a.com/b.js | 同一域名，不同端口 | 不允许 |
| http://www.a.com/a.js https://www.a.com/b.js | 同一域名，不同协议 | 不允许 |
| http://www.a.com/a.js http://70.32.92.74/b.js | 域名和域名对应ip | 不允许 |
| http://www.a.com/a.js http://script.a.com/b.js | 主域相同，子域不同 | 不允许 |
| http://www.a.com/a.js http://a.com/b.js | 同一域名，不同二级域名（同上） | 不允许（cookie这种情况下也不允许访问） |
| http://www.cnblogs.com/a.js http://www.a.com/b.js | 不同域名 | 不允许 |

特别注意两点：

第一，如果是协议和端口造成的跨域问题“前台”是无能为力的，

第二：在跨域问题上，域仅仅是通过“URL的首部”来识别而不会去尝试判断相同的ip地址对应着两个域或两个域是否在同一个ip上。

*“URL的首部”指window.location.protocol +window.location.host，也可以理解为“Domains, protocols and ports must match”。*

**2. 前端解决跨域问题**

**1>document.domain + iframe      (只有在主域相同的时候才能使用该方法)**

1) 在www.a.com/a.html中：

[复制代码](javascript:void(0);)

document.domain = 'a.com';

varifr = document.createElement('iframe');

ifr.src= 'http://www.script.a.com/b.html';

ifr.display= none;

document.body.appendChild(ifr);

ifr.onload= function(){

var doc = ifr.contentDocument ||ifr.contentWindow.document;

//在这里操作doc，也就是b.html

ifr.onload = null;

};

[复制代码](javascript:void(0);)

2) 在www.script.a.com/b.html中：

document.domain = 'a.com';

**2>动态创建script**

这个没什么好说的，因为script标签不受同源策略的限制。

[复制代码](javascript:void(0);)

functionloadScript(url, func) {

var head = document.head || document.getElementByTagName('head')[0];

var script = document.createElement('script');

script.src=url;

script.onload= script.onreadystatechange = function(){

if(!this.readyState || this.readyState=='loaded' || this.readyState=='complete'){

func();

script.onload= script.onreadystatechange = null;

}

};

head.insertBefore(script, 0);

}

window.baidu= {

sug: function(data){

console.log(data);

}

}

loadScript('http://suggestion.baidu.com/su?wd=w',function(){console.log('loaded')});

//我们请求的内容在哪里？

//我们可以在chorme调试面板的source中看到script引入的内容

[复制代码](javascript:void(0);)

**3>location.hash + iframe**

原理是利用location.hash来进行传值。

假设域名a.com下的文件cs1.html要和cnblogs.com域名下的cs2.html传递信息。  
1) cs1.html首先创建自动创建一个隐藏的iframe，iframe的src指向cnblogs.com域名下的cs2.html页面  
2) cs2.html响应请求后再将通过修改cs1.html的hash值来传递数据  
3) 同时在cs1.html上加一个定时器，隔一段时间来判断location.hash的值有没有变化，一旦有变化则获取获取hash值  
注：由于两个页面不在同一个域下IE、Chrome不允许修改parent.location.hash的值，所以要借助于a.com域名下的一个代理iframe  
  
代码如下：  
先是a.com下的文件cs1.html文件：

[复制代码](javascript:void(0);)

functionstartRequest(){

varifr = document.createElement('iframe');

ifr.style.display= 'none';

ifr.src= 'http://www.cnblogs.com/lab/cscript/cs2.html#paramdo';

document.body.appendChild(ifr);

}

functioncheckHash() {

try {

var data = location.hash ? location.hash.substring(1) : '';

if (console.log) {

console.log('Now the data is '+data);

}

} catch(e) {};

}

setInterval(checkHash, 2000);

[复制代码](javascript:void(0);)

cnblogs.com域名下的cs2.html:

[复制代码](javascript:void(0);)

//模拟一个简单的参数处理操作

switch(location.hash){

case '#paramdo':

callBack();

break;

case '#paramset':

//do something……

break;

}

functioncallBack(){

try {

parent.location.hash= 'somedata';

} catch (e) {

// ie、chrome的安全机制无法修改parent.location.hash，

// 所以要利用一个中间的cnblogs域下的代理iframe

varifrproxy = document.createElement('iframe');

ifrproxy.style.display= 'none';

ifrproxy.src= 'http://a.com/test/cscript/cs3.html#somedata'; // 注意该文件在"a.com"域下

document.body.appendChild(ifrproxy);

}

}

[复制代码](javascript:void(0);)

a.com下的域名cs3.html

//因为parent.parent和自身属于同一个域，所以可以改变其location.hash的值

parent.parent.location.hash = self.location.hash.substring(1);

**4> window.name + iframe**

window.name 的美妙之处：**name 值在不同的页面（甚至不同域名）加载后依旧存在，并且可以支持非常长的 name 值（2MB）。**

1) 创建a.com/cs1.html

2) 创建a.com/proxy.html，并加入如下代码

[复制代码](javascript:void(0);)

<head>

<script>

function proxy(url, func){

varisFirst = true,

ifr= document.createElement('iframe'),

loadFunc= function(){

if(isFirst){

ifr.contentWindow.location= 'http://a.com/cs1.html';

isFirst= false;

}else{

func(ifr.contentWindow.name);

ifr.contentWindow.close();

document.body.removeChild(ifr);

ifr.src= '';

ifr= null;

}

};

ifr.src=url;

ifr.style.display= 'none';

if(ifr.attachEvent) ifr.attachEvent('onload', loadFunc);

elseifr.onload =loadFunc;

document.body.appendChild(iframe);

}

</script>

</head>

<body>

<script>

proxy('http://www.baidu.com/', function(data){

console.log(data);

});

</script>

</body>

[复制代码](javascript:void(0);)

3 在b.com/cs1.html中包含：

<script>

window.name = '要传送的内容';

</script>

**5>postMessage（HTML5中的XMLHttpRequest Level 2中的API）**

1) a.com/index.html中的代码：

[复制代码](javascript:void(0);)

<iframe id="ifr" src="b.com/index.html"></iframe>

<script type="text/javascript">

window.onload= function() {

varifr = document.getElementById('ifr');

vartargetOrigin = 'http://b.com'; // 若写成'http://b.com/c/proxy.html'效果一样

// 若写成'http://c.com'就不会执行postMessage了

ifr.contentWindow.postMessage('I was there!', targetOrigin);

};

</script>

[复制代码](javascript:void(0);)

2) b.com/index.html中的代码：

[复制代码](javascript:void(0);)

<script type="text/javascript">

window.addEventListener('message', function(event){

// 通过origin属性判断消息来源地址

if (event.origin == 'http://a.com') {

alert(event.data); // 弹出"I was there!"

alert(event.source); // 对a.com、index.html中window对象的引用

// 但由于同源策略，这里event.source不可以访问window对象

}

}, false);

</script>

[复制代码](javascript:void(0);)

**6> CORS**

CORS背后的思想，就是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功，还是应该失败。

**IE中对CORS的实现是xdr**

[复制代码](javascript:void(0);)

varxdr = newXDomainRequest();

xdr.onload= function(){

console.log(xdr.responseText);

}

xdr.open('get', 'http://www.baidu.com');

......

xdr.send(null);

[复制代码](javascript:void(0);)

**其它浏览器中的实现就在xhr中**

[复制代码](javascript:void(0);)

varxhr = newXMLHttpRequest();

xhr.onreadystatechange= function () {

if(xhr.readyState == 4){

if(xhr.status>= 200 &&xhr.status< 304 || xhr.status == 304){

console.log(xhr.responseText);

}

}

}

xhr.open('get', 'http://www.baidu.com');

......

xhr.send(null);

[复制代码](javascript:void(0);)

**实现跨浏览器的CORS**

[复制代码](javascript:void(0);)

functioncreateCORS(method, url){

varxhr = newXMLHttpRequest();

if('withCredentials' inxhr){

xhr.open(method, url, true);

}elseif(typeofXDomainRequest != 'undefined'){

varxhr = newXDomainRequest();

xhr.open(method, url);

}else{

xhr= null;

}

returnxhr;

}

var request = createCORS('get', 'http://www.baidu.com');

if(request){

request.onload= function(){

......

};

request.send();

}

[复制代码](javascript:void(0);)

**7> JSONP**

JSONP包含两部分：回调函数和数据。

回调函数是当响应到来时要放在当前页面**被调用**的函数。

数据就是传入回调函数中的json数据，也就是回调函数的参数了。

[复制代码](javascript:void(0);)

functionhandleResponse(response){

console.log('The responsed data is: '+response.data);

}

var script = document.createElement('script');

script.src= 'http://www.baidu.com/json/?callback=handleResponse';

document.body.insertBefore(script, document.body.firstChild);

/\*handleResonse({"data": "zhe"})\*/

//原理如下：

//当我们通过script标签请求时

//后台就会根据相应的参数(json,handleResponse)

//来生成相应的json数据(handleResponse({"data": "zhe"}))

//最后这个返回的json数据(代码)就会被放在当前js文件中被执行

//至此跨域通信完成

[复制代码](javascript:void(0);)

 jsonp虽然很简单，但是有如下缺点：

1）安全问题(请求代码中可能存在安全隐患)

2）要确定jsonp请求是否失败并不容易

**8> web sockets**

web sockets是一种浏览器的API，它的目标是在一个单独的持久连接上提供全双工、双向通信。(同源策略对web sockets不适用)

web sockets原理：在JS创建了web socket之后，会有一个HTTP请求发送到浏览器以发起连接。取得服务器响应后，建立的连接会使用HTTP升级从HTTP协议交换为web sockt协议。

只有在支持web socket协议的服务器上才能正常工作。

var socket = newWebSockt('ws://www.baidu.com');//http->ws; https->wss

socket.send('hello WebSockt');

socket.onmessage= function(event){

var data =event.data;

}

分类: [JavaScript](http://www.cnblogs.com/JChen666/category/528065.html)